Django搭建简易博客教程

Andrew_liu

Published
with GitBook

Table of Contents

1. Introduce
2. Django简介
3. 开发环境和Django安装
4. 项目与APP
5. Models
6. Admin
7. Views和URL
8. Template
9. 动态URL
10. 多说,markdown和代码高亮

Introduction

本书主要面向 `Django学习者和博客开发者`，其中的代码军可以在作者的github上获得，主要是在Mac上进行开发，并没有在其他运行环境下做测试

希望达到的目标:

- 希望能写出一个系列文章，我也不知道到底能写多少
- 能够让认真阅读这个系列的文章的人，能在读完之后做出一个简单的博客
- 教会读者使用简单的git操作和github
- 希望能够加深自己对 `Django` 的理解

- 阅读手册
- Github地址
- 教程源代码地址
- 联系译者

`如发现任何不妥之处请联系作者，可直接发送邮件到liu.bin.coder@gmail.com或者直接对教程github源码进行Issue`

> 最后感谢阅读

django

Django简介

写作目的

喜欢一个学习观点`以教促学`, 一直以来, 学习的时候经常会发现, 某个方法某个问题自己已经明白了, 但是在教给别人的时候确说不清楚, 所以慢慢的学会了`以教促学`这种方法, 在教给别人知识的同时也能够提升自己对语言, 对框架的理解.

希望达到的目标:

- 希望能写出一个系列文章, 我也不知道到底能写多少
- 能够让认真阅读这个系列的文章的人, 能在读完之后做出一个简单的博客
- 教会读者使用简单的git操作和github
- 希望能够加深自己对`Django`的理解

Django简介

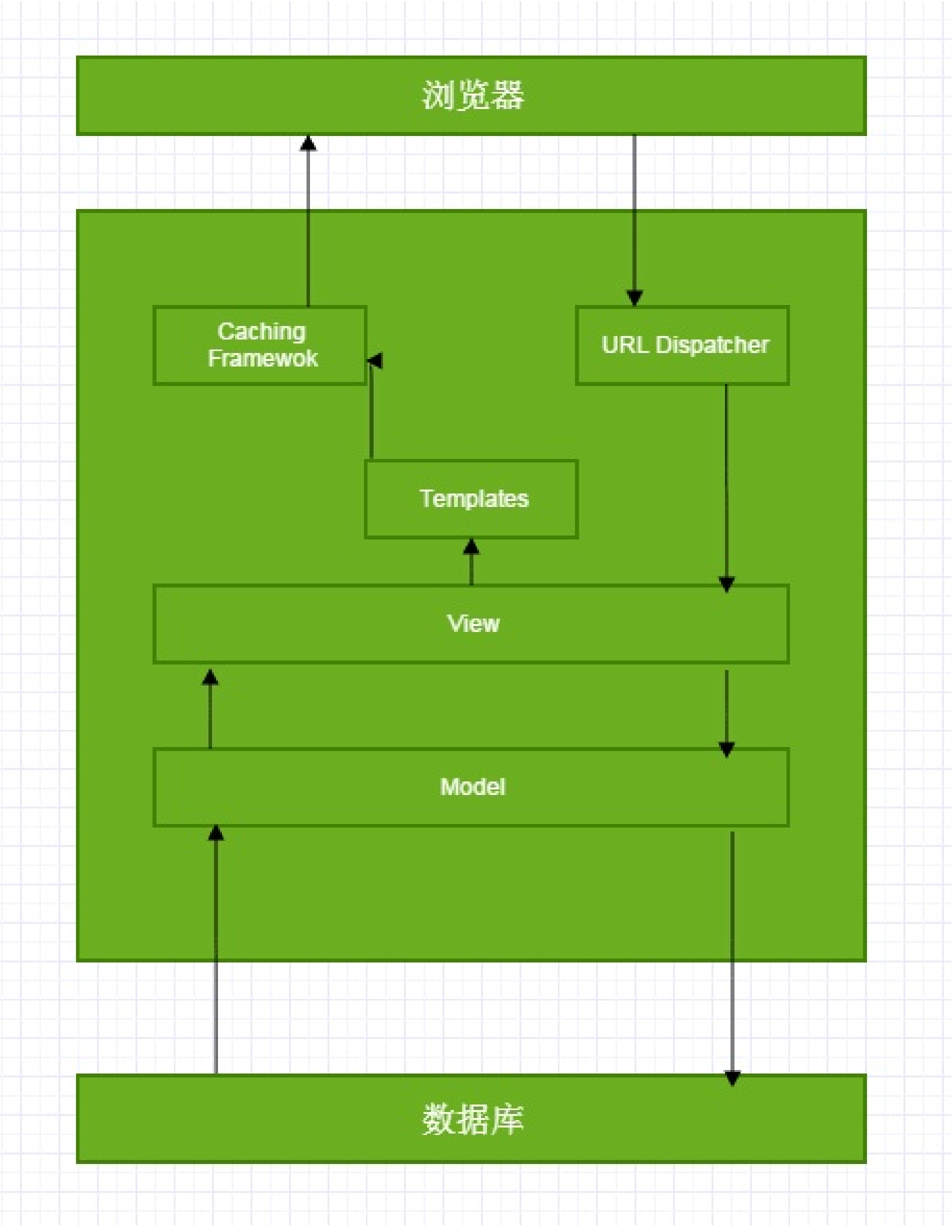
Django是`python`中目前风靡的Web Framework, 那么什么叫做`Framework`呢, 框架能够帮助你把程序的整体架构搭建好, 而我们所需要做的工作就是填写逻辑, 而框架能够在合适的时候调用你写的逻辑, 而不需要我们自己去调用逻辑, 让Web开发变的更敏捷.

> Django是一个高级Python Web框架, 鼓励快速,简洁, 以程序设计的思想进行开发. 通过使用这个框架, 可以减少很多开发麻烦, 使你更专注于编写自己的app, 而不需要重复造轮子. Django免费并且开源.

Django特点

- 完全免费并开源源代码
- 快速高效开发
- 使用MTV架构(`熟悉Web开发的应该会说是MVC架构`)
- 强大的可扩展性.

Django工作方式



用户在浏览器中输入`URL`后的回车, 浏览器会对`URL`进行检查, 首先判断协议,如果是`http`就按照 Web 来处理, 然互调用`DNS查询`, 将域名转换为`IP地址`, 然后经过网络传输到达对应Web服务器, 服务器对url进行解析后, 调用`View`中的逻辑(MTV中的V), 其中又涉及到Model(MTV中的M), 与数据库的进行交互, 将数据发到Template(MTV中的T)进行渲染, 然后发送到浏览器中, 浏览器以合适的方式呈现给用户

> 通过文字和图的结合希望读者能够初步理解Django的工作方式

开发环境和Django安装

开发环境

`下面仅仅是我的项目开发环境，没有必要追求完全一致...`

```
Mac OS X 10.10.1  #非必要
Python3.4.1
Django1.7.1
Bootstrap3.3.0 or Pure(临时决定使用的，@游逸 推荐) #非必要
Sublime Text 3  #非必要
virtualenv  1.11.6
```

虚拟环境配置

使用 `virtualenv` 创建虚拟环境, Ubuntun和Mac安装程序基本一致

```
#安装virtualenv
$ pip install virtualenv
#创建虚拟环境
$ virtualenv -p /usr/local/bin/python3.4 ENV3.4

Running virtualenv with interpreter /usr/local/bin/python3.4
Using base prefix '/Library/Frameworks/Python.framework/Versions/3.4'
New python executable in ENV3.4/bin/python3.4
Also creating executable in ENV3.4/bin/python
Installing setuptools, pip...done.

#激活虚拟环境
$ source /ENV3.4/bin/activate
#查看当前环境下的安装包
$ pip list
pip (1.5.6)
setuptools (3.6)
```

更多 `virtualenv` 使用可以参考Virtualenv简明教程

Git安装

Git是目前世界上最先进的分布式版本控制系统

Mac下git安装

```
$ brew install git
```

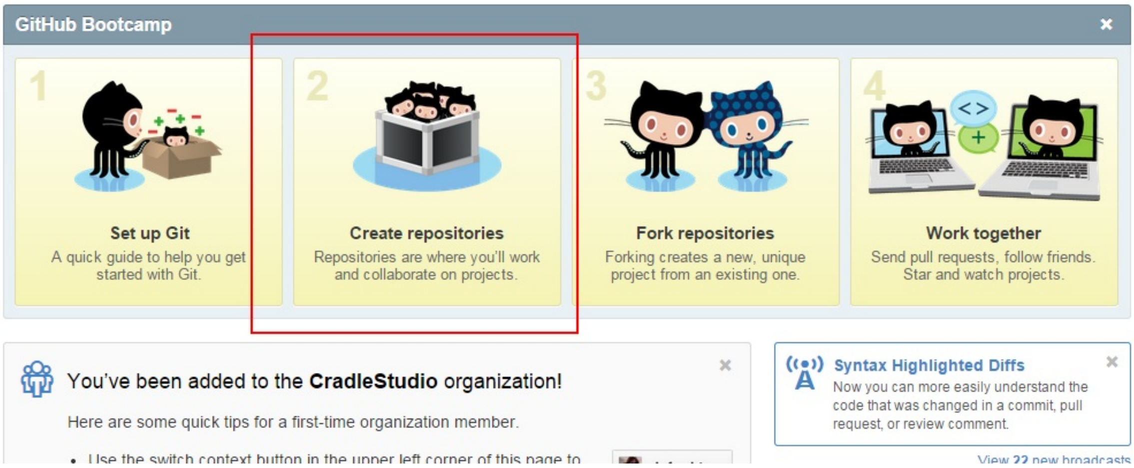
Ubuntu下git安装

```
$ sudo apt-get install git
```

Windows就不说了, 没怎么用过Windows做开发, 坑太多了

Github创建

在Github中创建一个属于自己的帐号 新建帐号后, 请点击 `New repository` 或者下图地方



并通过Install-SSH-Use-Github学习简单的Github与git的协作以及SSH的创建

> Github和git的协作我们会在使用的时候重复提示, 但最好先进行 `SSH的安装和配置`

Django安装

安装最新版的Django版本

```
#安装最新版本的Django
$ pip install  django
#或者指定安装版本
pip install -v django==1.7.1
```

Bootstrap安装

> `Bootstrap` 简洁、直观、强悍的前端开发框架，让web开发更迅速、简单

bootstrap已经有较为完善的中文文档, 可以在bootstrap中文网查看

推荐下载其中的Bootstrap源码

> 到目前为止, 基本环境已经搭建好了

项目与App

项目创建

现在正式开始吧, 我们创建一个名为 `my_blog` 的Django项目

创建项目的指令如下:

```
$ django-admin.py startproject my_blog
```

现在来看一下整个项目的文件结构

```
$ tree my_blog   #打印树形文件结构

my_blog
├── manage.py
└── my_blog
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py

1 directory, 5 files
```

建立Django app

> 在 `Django` 中的app我认为就是一个功能模块, 与其他的web框架可能有很大的区别, 将不能功能放在不同的app中, 方便代码的复用

建立一个 `article` app

```
$ python manage.py startapp article
```

现在让我们重新看一下整个项目的结构

```
── article
│   ├── __init__.py
│   ├── admin.py
│   ├── migrations
│   │   └── __init__.py
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── db.sqlite3
├── manage.py
├── my_blog
    ├── __init__.py
    ├── __pycache__
    │   ├── __init__.cpython-34.pyc
```

```
    │   ├── settings.cpython-34.pyc
    │   ├── urls.cpython-34.pyc
    │   └── wsgi.cpython-34.pyc
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

并在my_blog/my_blog/setting.py下添加新建app

```
INSTALLED_APPS = (
    ...
    'article',  #这里填写的是app的名称
)
```

运行程序

```
$ python manage.py runserver   #启动Django中的开发服务器
```

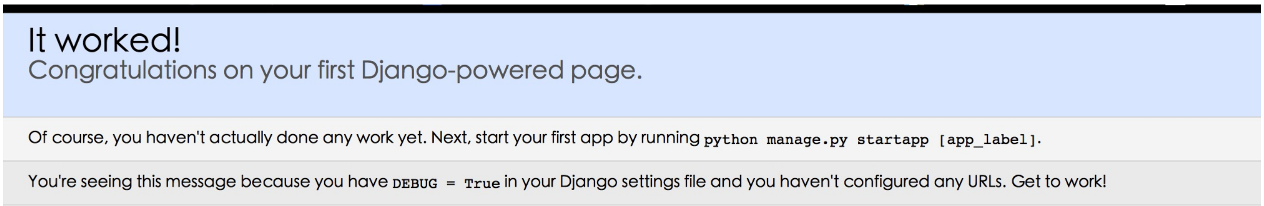
```
#如果运行上面命令出现以下提示
You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.
#请先使用下面命令
python manage.py migrate
#输出如下信息
Operations to perform:
  Apply all migrations: contenttypes, sessions, admin, auth
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying sessions.0001_initial... OK
```

运行成功后,会显示如下信息

```
#重新运行启动Django中的开发服务器
$ python manage.py runserver

#运行成功显示如下信息
System check identified no issues (0 silenced).
December 21, 2014 - 08:56:00
Django version 1.7.1, using settings 'my_blog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

现在可以启动浏览器, 输入http://127.0.0.1:8000/, 当出现



说明你成功走出了第一步!

命令梳理:

```
python manage.py <command> [options]  #Django Command python manange.py -h帮助文档
django-admin.py startproject my_blog  #创建项目
python manage.py startapp article  #创建app
```

Models

Django Model

- 每一个 `Django Model` 都继承自 `django.db.models.Model`
- 在 `Model` 当中每一个属性 `attribute` 都代表一个database field
- 通过 `Django Model API` 可以执行数据库的增删改查, 而不需要写一些数据库的查询语句

设置数据库

Django项目建成后, 默认设置了使用SQLite数据库, 在my_blog/my_blog/setting.py中可以查看和修改数据库设置:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

还可以设置其他数据库, 如 `MySQL, PostgreSQL` , 现在为了简单, 使用默认数据库设置

创建models

在my_blog/article/models.py下编写如下程序:

```
from django.db import models

# Create your models here.
class Article(models.Model) :
    title = models.CharField(max_length = 100)  #博客题目
    category = models.CharField(max_length = 50, blank = True)  #博客标签
    date_time = models.DateTimeField(auto_now_add = True)  #博客日期
    content = models.TextField(blank = True, null = True)  #博客文章正文

    def __unicode__(self) :
        return self.title

    class Meta:  #按时间下降排序
        ordering = ['-date_time']
```

其中 `__unicode__(self)` 函数Article对象要怎么表示自己, 一般系统默认使用 `<Article: Article object>` 来表示对象, 通过这个函数可以告诉系统使用title字段来表示这个对象

- `CharField` 用于存储字符串, max_length设置最大长度
- `TextField` 用于存储大量文本

- `DateTimeField` 用于存储时间, auto_now_add设置True表示自动设置对象增加时间

同步数据库

```
$ python manage.py migrate #命令行运行该命令
```

因为我们已经执行过该命令会出现如下提示

```
Operations to perform:
  Apply all migrations: admin, contenttypes, sessions, auth
Running migrations:
  No migrations to apply.
  Your models have changes that are not yet reflected in a migration, and so won't be app
  Run 'manage.py makemigrations' to make new migrations, and then re-run 'manage.py migra
```

那么现在需要执行下面的命令

```
$ python manage.py makemigrations
#得到如下提示
Migrations for 'article':
  0001_initial.py:
    - Create model Article
```

现在重新运行以下命令

```
$ python manage.py migrate
#出现如下提示表示操作成功
Operations to perform:
  Apply all migrations: auth, sessions, admin, article, contenttypes
Running migrations:
  Applying article.0001_initial... OK
```

> migrate命令按照app顺序建立或者更新数据库, 将`models.py`与数据库同步

Django Shell

现在我们进入Django中的交互式shell来进行数据库的增删改查等操作

```
$ python manage.py shell
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
```

> 这里进入Django的shell和python内置的shell是非常类似的

```
>>> from article.models import Article
>>> #create数据库增加操作
>>> Article.objects.create(title = 'Hello World', category = 'Python', content = '我们来做
<Article: Article object>
>>> Article.objects.create(title = 'Django Blog学习', category = 'Python', content = 'Djar
<Article: Article object>

>>> #all和get的数据库查看操作
>>> Article.objects.all()  #查看全部对象, 返回一个列表, 无对象返回空list
[<Article: Article object>, <Article: Article object>]
>>> Article.objects.get(id = 1)  #返回符合条件的对象
<Article: Article object>

>>> #update数据库修改操作
>>> first = Article.objects.get(id = 1)  #获取id = 1的对象
>>> first.title
'Hello World'
>>> first.date_time
datetime.datetime(2014, 12, 26, 13, 56, 48, 727425, tzinfo=<UTC>)
>>> first.content
'我们来做一个简单的数据库增加操作'
>>> first.category
'Python'
>>> first.content = 'Hello World, How are you'
>>> first.content  #再次查看是否修改成功, 修改操作就是点语法
'Hello World, How are you'

>>> #delete数据库删除操作
>>> first.delete()
>>> Article.objects.all()  #此时可以看到只有一个对象了, 另一个对象已经被成功删除
[<Article: Article object>]

Blog.objects.all()  # 选择全部对象
Blog.objects.filter(caption='blogname')  # 使用 filter() 按博客题目过滤
Blog.objects.filter(caption='blogname', id="1") # 也可以多个条件
#上面是精确匹配 也可以包含性查询
Blog.objects.filter(caption__contains='blogname')

Blog.objects.get(caption='blogname') # 获取单个对象 如果查询没有返回结果也会抛出异常

#数据排序
Blog.objects.order_by("caption")
Blog.objects.order_by("-caption")  # 倒序

#如果需要以多个字段为标准进行排序（第二个字段会在第一个字段的值相同的情况下被使用到），使用多个参数就可以了
Blog.objects.order_by("caption", "id")

#连锁查询
Blog.objects.filter(caption__contains='blogname').order_by("-id")

#限制返回的数据
Blog.objects.filter(caption__contains='blogname')[0]
Blog.objects.filter(caption__contains='blogname')[0:3]  # 可以进行类似于列表的操作
```

> 当然还有更多的API, 可以查看官方文档

Views和URL

网页程序的逻辑

> request进来->从服务器获取数据->处理数据->把网页呈现出来

- `url`设置相当于客户端向服务器发出request请求的`入口`, 并用来指明要调用的程序逻辑
- `views`用来处理程序逻辑, 然后呈现到template(一般为`GET`方法, `POST`方法略有不同)
- `template`一般为html+CSS的形式, 主要是呈现给用户的表现形式

简单Django Views和URL

Django中views里面的代码就是一个一个函数逻辑, 处理客户端(浏览器)发送的HTTPRequest, 然后返回HTTPResponse,

那么那么开始在my_blog/article/views.py中编写简单的逻辑

```
#现在你的views.py应该是这样
from django.shortcuts import render
from django.http import HttpResponse

# Create your views here.
def home(request):
    return HttpResponse("Hello World, Django")
```

那么如何使这个逻辑在http请求进入时, 被调用呢, 这里需要在`my_blog/my_blog/urls.py`中进行url设置

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'my_blog.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', 'article.views.home'),  #由于目前只有一个app, 方便起见, 就不设置include了
)
```

`url()`函数有四个参数, 两个是必须的:regex和view, 两个可选的:kwargs和name

- `regex`是regular expression的简写,这是字符串中的模式匹配的一种语法, Django 将请求的URL`从上至下依次匹配`列表中的正则表达式, 直到匹配到一个为止。

更多正则表达式的使用可以查看Python正则表达式

- `view`当 Django匹配了一个正则表达式就会调用指定的view逻辑, 上面代码中会调用article/views.py中

的home函数

- `kwargs` 任意关键字参数可传一个字典至目标view
- `name` 命名你的 URL, 使url在 Django 的其他地方使用, 特别是在模板中

现在在浏览器中输入127.0.0.1:8000应该可以看到下面的界面

Hello World, Django

Django Views和URL更近一步

很多时候我们希望给view中的函数逻辑传入参数, 从而呈现我们想要的结果

现在我们这样做, 在my_blog/article/views.py加入如下代码:

```
def detail(request, my_args):
    return HttpResponse("You're looking at my_args %s." % my_args)
```

在my_blog/my_blog/urls.py中设置对应的url,

```
urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'my_blog.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', 'article.views.home'),
    url(r'^(?P<my_args>\d+)/$', 'article.views.detail', name='detail'),
)
```

`^(?P<my_args>\d+)/$` 这个正则表达式的意思是将传入的一位或者多位数字作为参数传递到views中的detail作为参数, 其中 `?P<my_args>` 定义名称用于标识匹配的内容

一下url都能成功匹配这个正则表达数

- http://127.0.0.1:8000/1000/
- http://127.0.0.1:8000/9/

尝试传参访问数据库

修改在my_blog/article/views.py代码:

```
from django.shortcuts import render
from django.http import HttpResponse
from article.models import Article

# Create your views here.
def home(request):
    return HttpResponse("Hello World, Django")

def detail(request, my_args):
```

```
    post = Article.objects.all()[int(my_args)]
    str = ("title = %s, category = %s, date_time = %s, content = %s"
        % (post.title, post.category, post.date_time, post.content))
    return HttpResponse(str)
```

> 这里最好在admin后台管理界面增加几个Article对象, 防止查询对象为空, 出现异常

现在可以访问http://127.0.0.1:8000/1/

显示如下数据表示数据库访问正确(这些数据都是自己添加的), 并且注意Article.objects.all()返回的是一个列表

title = Hello World, category = python, date_time = 2014-12-27 02:12:04.773004+00:00, content = Hello Wordl

小结:

- 如何编写views和设置url
- 如何通过url向views传参
- 如何通过参数来访问数据库资源

Template

Template初探

到目前为止我们只是简单的将后端数据显示到页面上, 没有涉及到`HTML`代码, 而优雅的网站总算通过CSS+HTML, 甚至还有强大的JS的支持.

在这个教程中要打造一个Blog, 所以我们设置一个Blog界面, 原本打算使用`Bootstrap`作为前段的工具, 不过经过`@游逸`的建议, 使用了更加轻量级的Pure, 同样是响应式页面设置, 这也将是未来的主流吧..

在my_blog下添加文件名, 文件夹名为`templates`

```
mkdir templates
#看到当前文件构成
my_blog
├── article
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-34.pyc
│   │   ├── admin.cpython-34.pyc
│   │   ├── models.cpython-34.pyc
│   │   └── views.cpython-34.pyc
│   ├── admin.py
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   │       ├── 0001_initial.cpython-34.pyc
│   │       └── __init__.cpython-34.pyc
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── db.sqlite3
├── manage.py
├── my_blog
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-34.pyc
│   │   ├── settings.cpython-34.pyc
│   │   ├── urls.cpython-34.pyc
│   │   └── wsgi.cpython-34.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── templates
```

在my_blog/my_blog/setting.py下设置templates的位置

```
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates').replace('\\', '/'),
    )
```

意思是告知项目templates文件夹在项目根目录下

第一个template

---

templates/test.html简单第一个 template html文件

```
<!--在test.html文件夹下添加-->
<!DOCTYPE html>
<html>
    <head>
        <title>Just test template</title>
        <style>
            body {
               background-color: red;
            }
            em {
                color: LightSeaGreen;
            }
        </style>
    </head>
    <body>
        <h1>Hello World!</h1>
        <strong>{{ current_time }}</strong>
    </body>
</html>
```

其中`{{ current_time }}`是Django Template中变量的表示方式

在article/view.py中添加一个函数逻辑

```
from django.shortcuts import render
from django.http import HttpResponse
from article.models import Article
from datetime import datetime

# Create your views here.
def home(request):
    return HttpResponse("Hello World, Django")

def detail(request, my_args):
    post = Article.objects.all()[int(my_args)]
    str = ("title = %s, category = %s, date_time = %s, content = %s"
        % (post.title, post.category, post.date_time, post.content))
    return HttpResponse(str)

def test(request) :
    return render(request, 'test.html', {'current_time': datetime.now()})
```

`render()`函数中第一个参数是`request` 对象, 第二个参数是一个`模板名称`， 第三个是一个`字典类型`的可选参数. 它将返回一个包含有给定模板根据给定的上下文渲染结果的 `HttpResponse` 对象。

然后设置对应的url在my_blog/urls.py下

```
    url(r'^test/$', 'article.views.test'),
```

重新启动服务器`python manage.py runserver`, 然后在浏览器中输入http://127.0.0.1:8000/test/, 可以看到

正式编写template

在template文件夹下增加base.html, 并在其中增加如下代码

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="description" content="A layout example that shows off a blog page with a list

    <title>Andrew Liu Blog</title>
    <link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.5.0/pure-min.css">
    <link rel="stylesheet" href="http://yui.yahooapis.com/pure/0.5.0/grids-responsive-min
    <link rel="stylesheet" href="http://picturebag.qiniudn.com/blog.css">
</head>
<body>
<div id="layout" class="pure-g">
    <div class="sidebar pure-u-1 pure-u-md-1-4">
        <div class="header">
            <h1 class="brand-title">Andrew Liu Blog</h1>
            <h2 class="brand-tagline">雪忆 - Snow Memory</h2>
            <nav class="nav">
                <ul class="nav-list">
                    <li class="nav-item">
                        <a class="pure-button" href="https://github.com/Andrew-liu">Githu
                    </li>
                    <li class="nav-item">
                        <a class="pure-button" href="http://weibo.com/dinosaurliu">Weibo<
                    </li>
                </ul>
            </nav>
        </div>
    </div>

    <div class="content pure-u-1 pure-u-md-3-4">
        <div>
            {% block content %}
            {% endblock %}
            <div class="footer">
```

```
                  <div class="pure-menu pure-menu-horizontal pure-menu-open">
                      <ul>
                          <li><a href="http://andrewliu.tk/about/">About Me</a></li>
                          <li><a href="http://twitter.com/yuilibrary/">Twitter</a></li>
                          <li><a href="http://github.com/yahoo/pure/">GitHub</a></li>
                      </ul>
                  </div>
              </div>
          </div>
      </div>
  </div>

  </body>
  </html>
```

上面这段html编写的页面是一个模板, 其中`{% block content %} {% endblock %}`字段用来被其他继承这个基类模板进行重写

我们继续在templates文件夹下添加home.html文件

```
{% extends "base.html" %}

{% block content %}
<div class="posts">
    {% for post in post_list %}
        <section class="post">
            <header class="post-header">
                <h2 class="post-title">{{ post.title }}</h2>

                    <p class="post-meta">
                        Time:  <a class="post-author" href="#">{{ post.date_time }}</a> <
                    </p>
            </header>

                <div class="post-description">
                    <p>
                        {{ post.content }}
                    </p>
                </div>
        </section>
    {% endfor %}
</div><!-- /.blog-post -->
{% endblock %}
```

其中

- `{% for <element> in <list> %}与{% endfor %}`成对存在, 这是template中提供的for循环tag
- `{% if <elemtnt> %} {% else %} {% endif %}`是template中提供的if语句tag
- template中还提供了一些过滤器

然后修改my_blog/article/view.py, 并删除test.html

```
# -*- coding: utf-8 -*-
from django.shortcuts import render
from django.http import HttpResponse
```

```
from article.models import Article
from datetime import datetime

# Create your views here.
def home(request):
    post_list = Article.objects.all()  #获取全部的Article对象
    return render(request, 'home.html', {'post_list' : post_list})
```

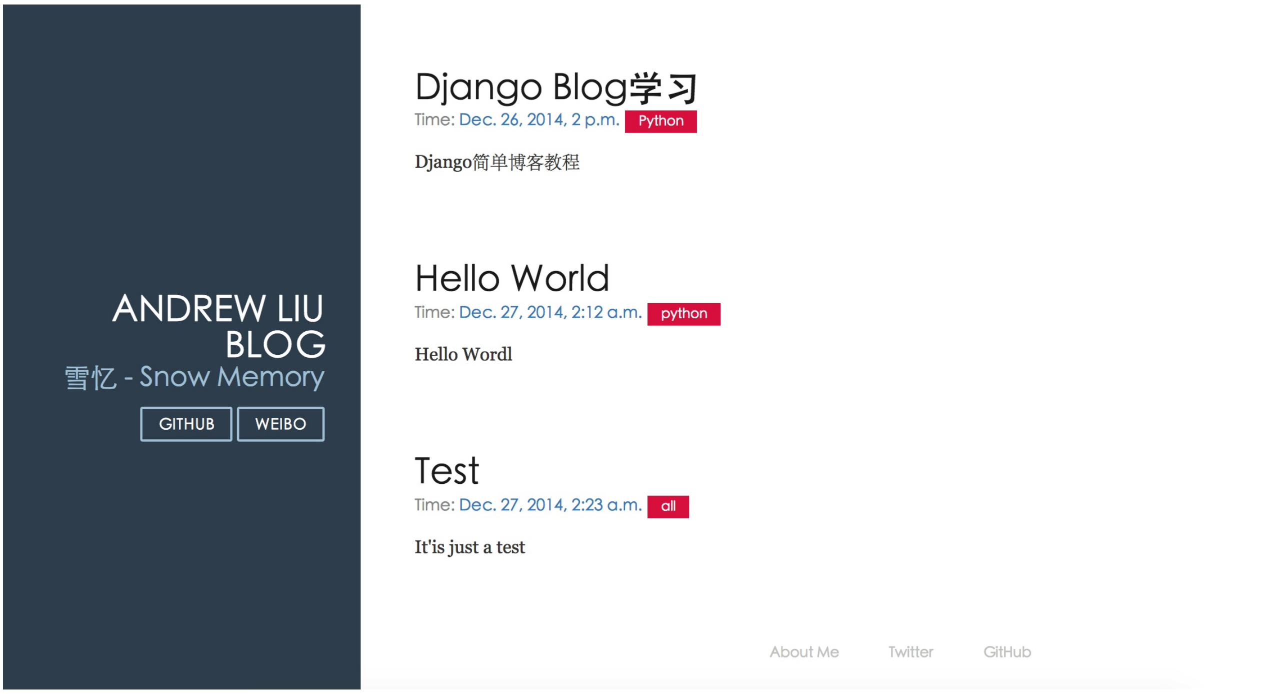
修改my_blog/my_blog/urls.py

```
from django.conf.urls import patterns, include, url
from django.contrib import admin

urlpatterns = patterns('',
    # Examples:
    # url(r'^$', 'my_blog.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', 'article.views.home'),
)
```

现在重新打开http://127.0.0.1:8000/, 发现Blog的整理框架已经基本完成, 到现在我们已经了解了一些Django的基本知识, 搭建了简单地Blog框架, 剩下的就是给Blog添加功能



查看当前整个程序的目录结构

```
my_blog
├── article
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-34.pyc
│   │   ├── admin.cpython-34.pyc
│   │   ├── models.cpython-34.pyc
│   │   └── views.cpython-34.pyc
│   ├── admin.py
```

```
│   ├── migrations
│   │   ├── 0001_initial.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   │       ├── 0001_initial.cpython-34.pyc
│   │       └── __init__.cpython-34.pyc
│   ├── models.py
│   ├── tests.py
│   └── views.py
├── db.sqlite3
├── manage.py
├── my_blog
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-34.pyc
│   │   ├── settings.cpython-34.pyc
│   │   ├── urls.cpython-34.pyc
│   │   └── wsgi.cpython-34.pyc
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
└── templates
    ├── base.html
    └── home.html
```

将代码上传到Github

在github中新建仓库`my_blog_tutorial`，填写简单的描述

```
#查看当前目录位置
$ pwd
/Users/andrew_liu/Python/Django/my_blog

#在项目的根目录下初始化git
git init
Initialized empty Git repository in/Users/andrew_liu/Python/Django/my_blog/.git/

#添加远程github
$ git remote add blog git@github.com:Andrew-liu/my_blog_tutorial.git
```

在根目录下增加`.gitignore'和'LICENSE'和'README.md'文件

```
#添加所有文件
$ git add .

#查看当前状态
$ git status

#commit操作
$ git commit -m "django tutorial init"

#上传github
$ git push -u blog master
Counting objects: 23, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (22/22), done.
```

```
Writing objects: 100% (23/23), 19.56 KiB | 0 bytes/s, done.
Total 23 (delta 1), reused 0 (delta 0)
To git@github.com:Andrew-liu/my_blog_tutorial.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from blog.
```

动态URL

多说,markdown和代码高亮